TP Qt : QCM

© 2012 tv <tvaira@free.fr> - v.1.0 - le 8 mars 2012

Sommaire

Manipulations **2**

Objectifs . . . 2

Mise en situation . . . 2

Travail demandé **5**

Itération 1 . . . 5

Itération 2 . . . 10

Itération 3 . . . 12

Itération 4 . . . 16

Liens . . . 18

Listings

1 QCM.h . . . 3

2 creer examen() . . . 4

3 main.cpp . . . 5

4 QCMDialog.h . . . 8

5 QCMDialog.cpp . . . 9

6 Exemple de fichier CSV produit . . . 10

7 Ecriture au format CSV . . . 11

8 main3.cpp . . . 12

9 Exemple de fichier XML . . . 12

10 Exemple d'ouverture d'un fichier XML . . . 13

11 Exemple de parcours d'un fichier XML . . . 14

12 Exemple de parcours d'une liste de noeuds . . . 14

13 Exemple d'accès à un élément de la liste . . . 14

Manipulations

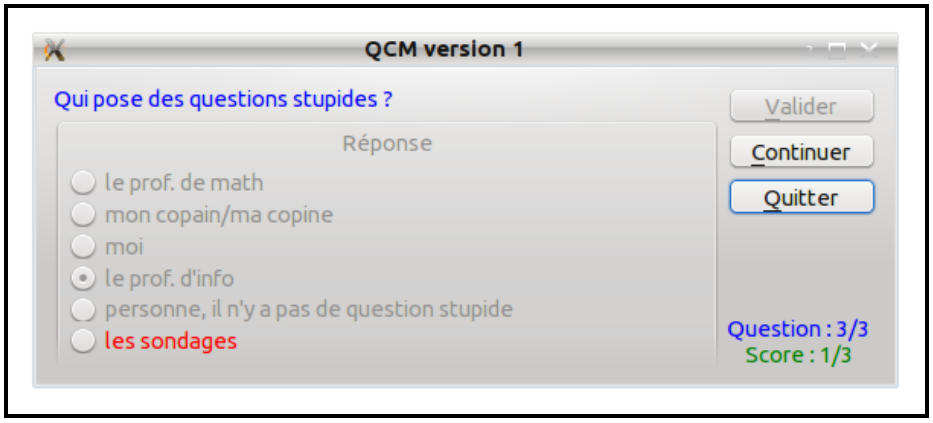
Objectifs

Les objectifs de ce TP sont :

- créer des boîtes de dialogue personnalisées et utiliser les boîtes de Qt
- manipuler les types structurés et vector de la STL
- gérer des fichiers au format CSV (*Comma-Separated Values*) et XML (*Extensible Markup Language*)
- utiliser les fonctions de dessin de `QPainter`

Mise en situation

Dans ce TP, il s'agit de réaliser un programme d'examen sous forme de questionnaire à choix multiple (QCM) où une question est posée et la réponse est à choisir parmi un ensemble de propositions.



La version 1

On va développer l'application en 4 itérations :

- itération n°1 : la réalisation de la GUI (*Graphical User Interface*)
- itération n°2 : la sauvegarde du résultat au format CSV
- itération n°3 : la lecture d'un questionnaire au format XML
- itération n°4 : une boîte de lancement permettant de choisir l'examen et l'affichage d'un chronomètre

Remarque : un développement itératif s'organise en une série de développement très courts de durée fixe nommée itérations. Le résultat de chaque itération est un système partiel exécutable, testé et intégré (mais incomplet).

On tiendra compte des contraintes initiales suivantes :

- une seule bonne réponse possible par question
- les questions n'auront pas toutes le même nombre de propositions

 © 2012 tv <tvaira@free.fr>

On définit une structure `QCM` comprenant quatre champs :

- un champ **question** (chaîne de caractères) qui contiendra la question à poser
- un champ **reponses** (tableau de taille variable de chaîne de caractères) contenant les propositions
- un champ **solution** (entier) qui contient le numéro de la bonne réponse (dans le champ reponses)
- un champ **point** (entier) qui attribue un nombre de points pour avoir trouver la bonne réponse

On cherche maintenant à faire un examen de plusieurs questions. On définit pour cela un type `Examen` comme un tableau dynamique de structures `QCM` en utilisant le type `vector` de la STL.

Ces définitions sont regroupées dans le fichier *header* `QCM.h` fourni.

```
#ifndef QCM_H
#define QCM_H

#include <iostream>
#include <QString>
#include <vector>

using namespace std;

struct QCM
{
  QString question;
  vector<QString> reponses;
  unsigned int solution;
  unsigned int point;
};

typedef vector<QCM> Examen;

Examen creer_examen();

#endif // QCM_H
```

Code 1: QCM.h

 © 2012 tv <tvaira@free.fr>

Pour créer un `Examen` (un ensemble de questions à choix multiple et pour nous un `vector` de structures `QCM`), on utilisera dans le `main.cpp` une fonction `creer_examen()` (seulement pour les itérations 1 et 2) :

```
Examen creer_examen()
{
  QCM q;
  Examen _examen;

  q.question = QString::fromUtf8("Laquelle␣des␣expressions␣suivantes␣est␣un␣prototype␣de␣
     fonction␣?");
  q.reponses.clear();
  q.reponses.push_back("int␣f(0);");
  q.reponses.push_back("int␣f(int␣0);");
  q.reponses.push_back("int␣f(int␣i);");
  q.reponses.push_back("int␣f(i);");
  q.solution=3;
  q.point=1;
  _examen.push_back(q);

  q.question = QString::fromUtf8("Qui␣pose␣des␣questions␣stupides␣?");
  q.reponses.clear();
  q.reponses.push_back("le␣prof.␣de␣math");
  q.reponses.push_back("mon␣copain/ma␣copine");
  q.reponses.push_back("moi");
  q.reponses.push_back("le␣prof.␣d'info");
  q.reponses.push_back("personne,␣il␣n'y␣a␣pas␣de␣question␣stupide");
  q.reponses.push_back("les␣sondages");
  q.solution=6;
  q.point=1;
  _examen.push_back(q);

  /* etc ... */

  return _examen;
}
```

Code 2: creer examen()

Documentation "STL Containers : vector" : www.cplusplus.com/reference/stl/vector/

Travail demandé

Itération 1

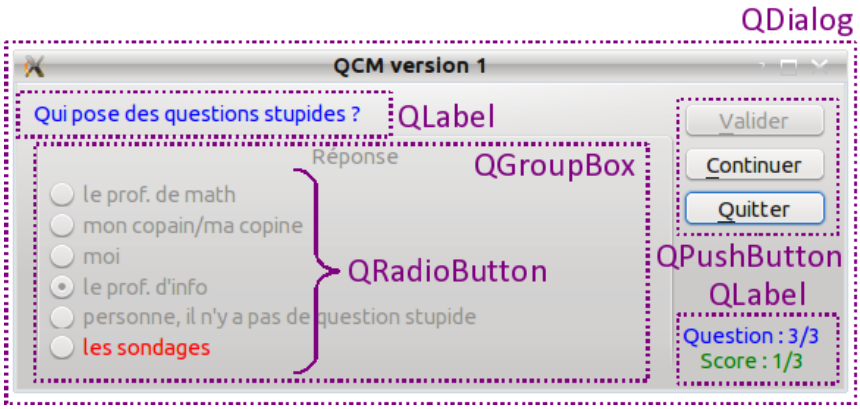
Dans cette première itération, on s'attachera à créer sa propre boîte de dialogue en créant une nouvelle classe **QCMDialog** qui héritera de la classe **QDialog**. Une instance de cette classe représentera (pour l'instant) la fenêtre de l'application :

```
#include <QtGui>
#include "QCMDialog.h"
#include "QCM.h"

using namespace std;

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    Examen examen = creer_examen();

    QCMDialog dialog(examen);
    dialog.show();

    return app.exec();
}

Examen creer_examen()
{
...
}
```

Code 3: main.cpp

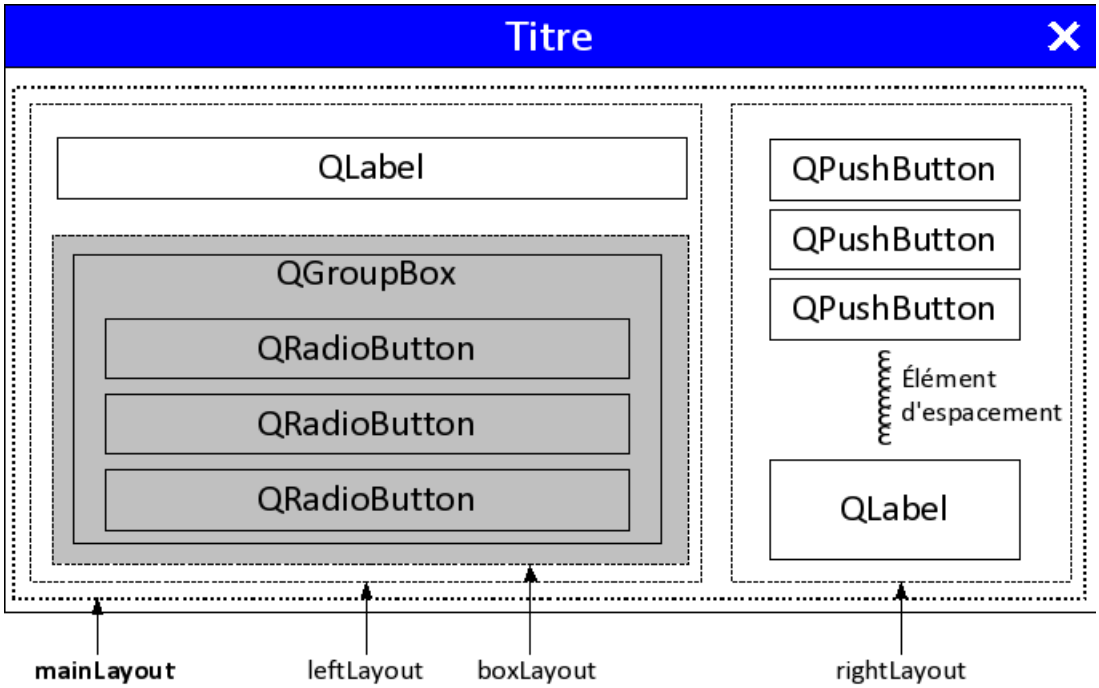
Le code source de cette classe sera réparti en deux fichiers : `QCMDialog.h` et `QCMDialog.cpp`.

Cette boîte de dialogue comprendra les widgets suivants : deux `QLabel`, un `QGroupBox` qui contiendra n `QRadioButton` et trois `QPushButton`.



 © 2012 tv <tvaira@free.fr>

Le positionnement de ces widgets respectera le plan suivant :



L'instanciation des widgets et leur positionnement se fera en partie dans le constructeur de la classe `QCMDialog` (voir Code 5). Les éléments d'espacement s'obtiennent en appelant la méthode `addStretch()` du layout concerné.

Pour assurer ce positionnement, il vous faudra utiliser le principe des conteneurs *layout* offert par Qt pour organiser les différents widgets dans la boîte de dialogue. Voici les quatre *layout* à mettre en oeuvre : mainLayout → QHBoxLayout, leftLayout → QVBoxLayout, rightLayout → QVBoxLayout et boxLayout → QVBoxLayout.

La classe `QCMDialog` devra utiliser la macro `Q_OBJECT` nécessaire pour toutes les classes qui définissent des signaux et (/ou) des slots.

Le rôle des trois boutons est le suivant :

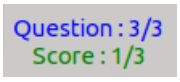
- le bouton Valider valide la proposition choisie et déclenche l'évaluation de la réponse fournie
- le bouton Continuer déclenche le passage à la question suivante ou, si c'est la dernière question, la fin du QCM en affichant le résultat obtenu à ce questionnaire
- le bouton Quitter permet de terminer l'application

Cette boîte de dialogue gérera trois slots privés :

- `enableValiderButton()` qui est appelé dès qu'il y aura un clic sur un des boutons Radio et qui permettra d'activer le bouton Valider si une proposition a été choisie
- `continuerClicked()` qui est invoqué lorsque l'utilisateur clique sur le bouton Continuer
- `validerClicked()` qui est invoqué lorsque l'utilisateur clique sur le bouton Valider

Le bouton Valider sera le bouton par défaut (c'est-à-dire celui qui est pressé quand l'utilisateur appuie sur la touche Entrée). Il sera aussi désactivé par défaut. La boîte de dialogue sera fermée lorsque l'utilisateur cliquera sur le bouton Quitter.

Une zone de la boîte de dialogue est réservée au suivi du questionnaire en réalisant :

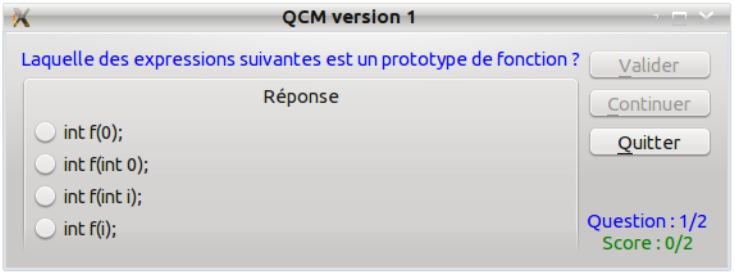


- l'affichage (en bleu) du numéro de la question courante sur le nombre total de questions
- l'affichage (en vert) du nombre de points obtenus sur le total possible pour ce QCM

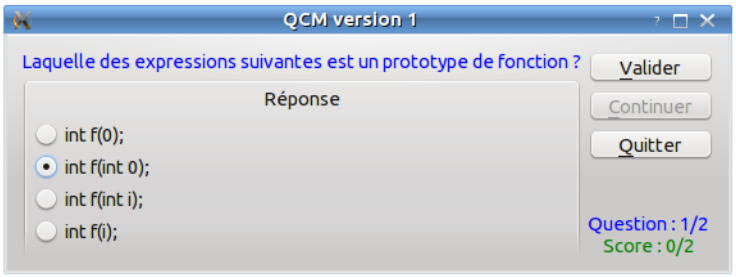
Pour personnaliser légèrement les affichages dans la boîte de dialogue, la question sera affichée en bleu et la bonne réponse en rouge.

Le manuel d'utilisation pour cette version est décrit ci-dessous :

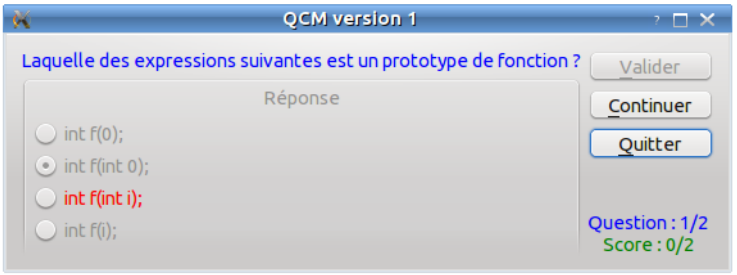
La boîte de dialogue apparaît en affichant la première question de l'Examen →



L'utilisateur doit choisir une proposition (ou sinon quitter l'application) →

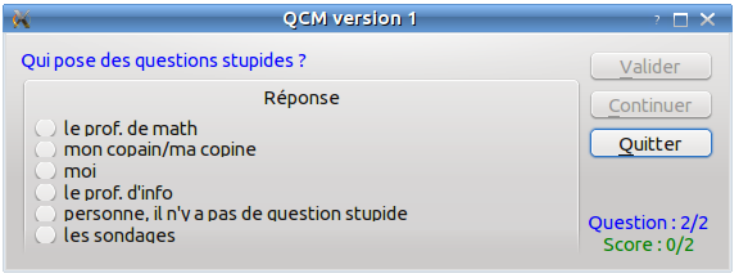


Il peut maintenant valider son choix et obtenir le résultat à cette question →

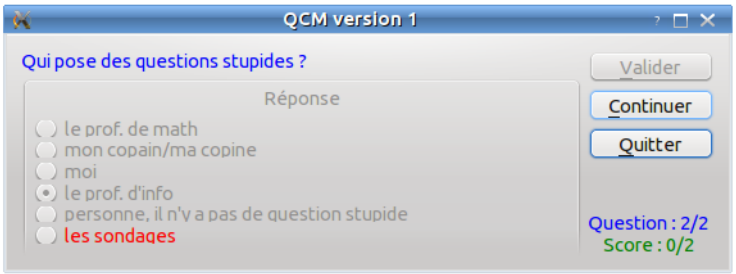


Il clique sur Continuer pour afficher la question suivante →

L'utilisateur choisit une proposition et valide son choix ...

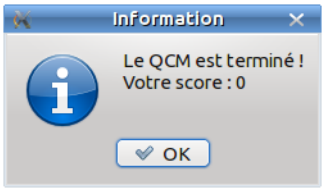


La bonne réponse s'affiche et son score est mis à jour →

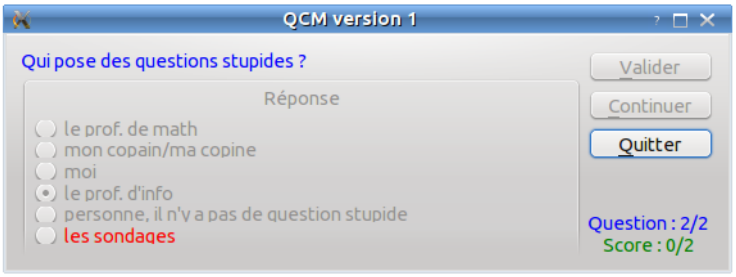


 © 2012 tv <tvaira@free.fr>

Il clique sur Continuer et le résultat obtenu s'affiche car c'était la dernière question →



Ce questionnaire est maintenant terminé, il peut Quitter l'application →



Le constructeur de la classe **QCMDialog** a la charge d'initialiser les attributs et d'afficher la première question du QCM. La seule difficulté à prendre en compte est la contrainte suivante : les questions n'auront pas toutes le même nombre de propositions. Cela implique que le nombre de `QRadioButton` à instancier va varier en fonction de la question. Par contre les autres widgets sont communs à toutes les questions et pourront être conservés pour chaque affichage. Seuls les contenus des `QLabel` et l'état des boutons seront mis à jour.

Il y a plusieurs façons de gérer un nombre variable d'objets :

1. un tableau dynamique de pointeurs sur des objets de type `QRadioButton`
2. un conteneur d'objets de type `QRadioButton`

Comme la STL, Qt fournit un ensemble de conteneur prêt à l'emploi : `QList<T>`, `QLinkedList<T>`, `QQueue<T>`, `QVector<T>`, `QMap<Key, T>` ... Vous êtes libre d'utiliser la technique qui vous convient.

Question 1. Compléter la déclaration de la classe **QCMDialog**.

```
#ifndef MYWIDGET_H
#define MYWIDGET_H

#include <QtGui>

class QCMDialog : public QDialog
{
    Q_OBJECT
    private:
        /* ... */

    public:
        QCMDialog(const Examen& examen, QDialog *parent = 0);
        ~QCMDialog();

    private slots:
        /* les slots validerClicked, continuerClicked et enableValiderButton */
};
#endif
```

Code 4: QCMDialog.h

Question 2. Compléter la définition de la classe **`QCMDialog`**.

```
#include <QtGui>

#include "QCMDialog.h"

QCMDialog::QCMDialog(const Examen& examen, QDialog *parent) : QDialog(parent)
{
  /* TODO */
  ...

  setWindowTitle(tr("QCM␣version␣1"));
}

QCMDialog::~QCMDialog()
{}

void QCMDialog::validerClicked()
{
  /* TODO */
}

void QCMDialog::continuerClicked()
{
  /* TODO */
}

void QCMDialog::enableValiderButton(bool checked)
{
  /* TODO */
}

...
```

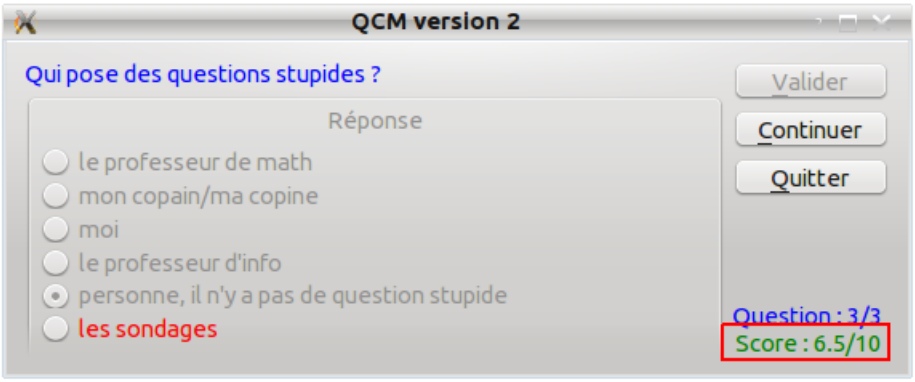
Code 5: QCMDialog.cpp

Question 3. Fabriquer l'application version 1 et tester.

 © 2012 tv <tvaira@free.fr>

Itération 2

Cette version intègre une modification de la version 1 et un ajout de fonctionnalité.



Il serait intéressant de pouvoir gérer des demi-points (0,5) dans le barème des questions. Vous devez assurer les modifications pour prendre en compte cette demande.

On désire pouvoir récupérer un fichier (au format CSV) contenant les résultats obtenus par l'utilisateur aux questions posées. Cela permettra d'exploiter ce fichier dans un logiciel de type "tableur".

	A	B	C	D
1	Question	Reponse	Point	Score
2	1	2	3	0
3	2	6	3,5	3,5
4				
5				3,5

Remarque : si vous ne connaissez pas ce format, vous devez lire au moins l'article disponible sur Wikipedia (fr.wikipedia.org/wiki/Comma-separated_values).

On enregistrera tout au long du QCM les informations suivantes :

- colonne **Question** : le numéro de la question
- colonne **Réponse** : le numéro de la proposition choisie
- colonne **Point** : le nombre de point affecté à cette question
- colonne **Score** : le nombre de point obtenu par l'utilisateur, puis une fois le QCM terminé, on ajoutera le total obtenu (dans la colonne **Score**).

On prendra comme délimiteur de champ le **point-virgule** (;) et on utilisera la virgule (,) pour représenter les nombres décimaux (standard français).

Pour mettre en oeuvre cet ajout de fonctionnalité, vous utiliserez la classe **`QFile`** fournie par Qt. Comme les fichiers CSV sont de "simples" fichiers textes, vous pouvez utiliser la classe **`QTextStream`** pour écrire un flux dans le fichier (`<<`).

```
Question;Reponse;Point;Score;
1;2;3,00;0,00;
2;6;3,50;3,50;

;;;3,50;
```

Code 6: Exemple de fichier CSV produit

Exemple d'écriture dans un fichier "texte" :

```
outfile = new QFile("sample.csv");
if (outfile->open(QFile::WriteOnly | QIODevice::Text))
{
    QTextStream out(outfile);
    out << "Question;Reponse;Point;Score;" << endl;
}
outfile->close();
```

Code 7: Ecriture au format CSV

Documentation `QFile` : developer.qt.nokia.com/doc/qfile.html

Documentation `QTextStream` : developer.qt.nokia.com/doc/qtextstream.html

Question 4. Modifier l'application pour gérer les demi-points.

Question 5. Ajouter la fonctionnalité d'enregistrement des résultats au format CSV.

Question 6. Fabriquer l'application version 2 et tester.

 © 2012 tv <tvaira@free.fr>

Itération 3

On va maintenant doter le programme de la faculté de lire depuis un fichier xml le QCM à soumettre à l'utilisateur.

On modifiera le main de l'application afin de passer en argument de la classe `QCMDialog` le nom du fichier contenant l'examen :

```
#include <QtGui>
#include "QCMDialog.h"
#include "QCM.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    QCMDialog dialog("qcm-2"); /* correspond au fichier qcm-2.xml */
    dialog.show();

    return app.exec();
}
```

Code 8: main3.cpp

Le format d'un fichier XML à lire est le suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<questions>
   <question>
      <libelle>Laquelle des expressions suivantes est un prototype de fonction ?</libelle>
      <propositions>
         <proposition libelle="int␣f(0);" />
         <proposition libelle="int␣f(int␣0);" />
         <proposition libelle="int␣f(int␣i);" />
         <proposition libelle="int␣f(i);" />
      </propositions>
      <solution>3</solution>
      <point>1.5</point>
   </question>
   <question>
      <libelle>Qui pose des questions stupides ?</libelle>
      <propositions>
         <proposition libelle="le␣professeur␣de␣math" />
         <proposition libelle="mon␣copain/ma␣copine" />
         <proposition libelle="moi" />
         <proposition libelle="le␣professeur␣d'info" />
         <proposition libelle="personne,␣il␣n'y␣a␣pas␣de␣question␣stupide" />
         <proposition libelle="les␣sondages" />
      </propositions>
      <solution>6</solution>
      <point>1</point>
   </question>
</questions>
```

Code 9: Exemple de fichier XML

Remarque : si vous ne connaissez pas ce format, vous devez lire au moins l'article disponible sur Wikipedia (fr.wikipedia.org/wiki/Xml).

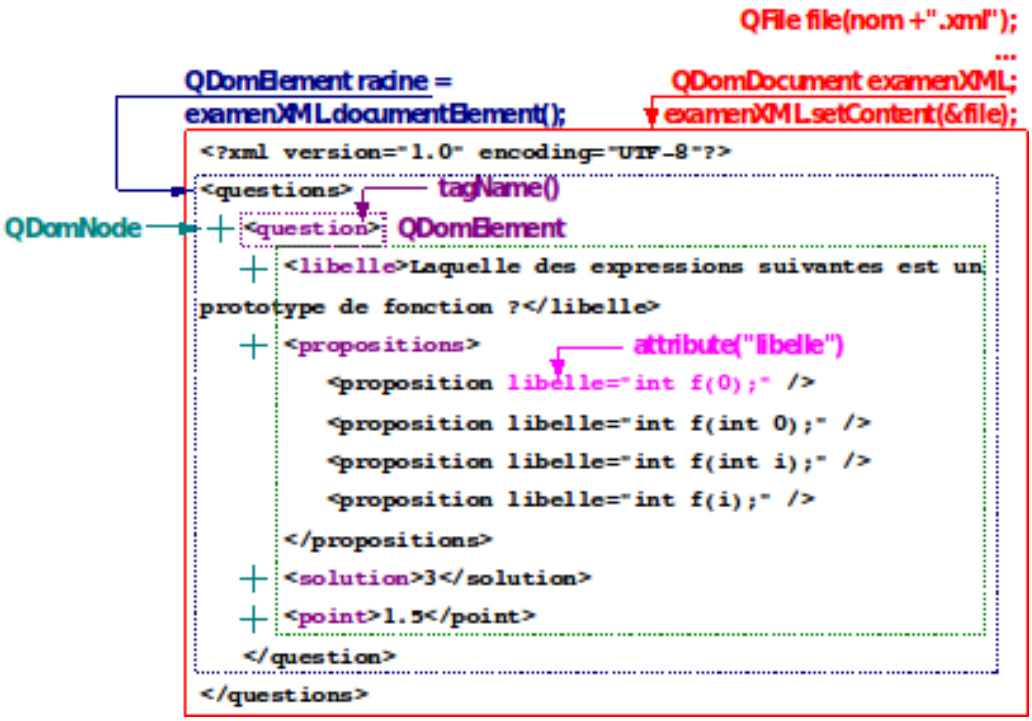
La première modification à apporter se trouve dans le fichier de projet (.pro) en ajoutant le support **xml**. Pour cela, il faut ajouter cette ligne :

```
QT += xml
```

Maintenant, il faut ajouter une méthode privée à la classe `QCMDialog` : **`bool lireExamen()`**. Cette méthode va lire et décoder le fichier xml et créer un Examen. Elle remplace la fonction `creer_examen()` utilisée jusqu'à maintenant.

Le support **xml** de Qt fournit de nombreuses classes pour le travail à effectuer ici : `QDomDocument` qui représente un document XML, `QDomElement` qui représente un élément de l'arbre DOM (*Document Object Model*), `QDomNode` et `QDomNodeList` qui représentent un noeud et une liste de noeuds, ... Il est fortement conseillé de lire la documentation Qt à ce sujet (developer.qt.nokia.com/doc/xml-processing.html).

Exemple de décomposition d'un document XML :



Sous Qt, l'ouverture d'un fichier XML sera réalisé de la manière suivante :

```
QString nom("examen-1");
QFile file(nom + ".xml");
QDomDocument examenXML;

if (!file.open(QIODevice::ReadOnly))
{
  QMessageBox::critical(this,"Erreur",QString::fromUtf8("Le␣fichier␣") + nom + QString::
     fromUtf8(".xml␣n'est␣pas␣accessible␣!"),QMessageBox::Ok,0);
  return false;
}
```

```
if (!examenXML.setContent(&file))
{
  file.close();
  QMessageBox::critical(this,"Erreur",QString::fromUtf8("Le␣fichier␣") + nom + QString::
     fromUtf8(".xml␣n'est␣pas␣valide␣!"),QMessageBox::Ok,0);
  return false;
}
file.close();
```

Code 10: Exemple d'ouverture d'un fichier XML

Ensuite, on parcourt le document XML en accédant successivement aux noeuds le composant :

```
QDomElement racine = examenXML.documentElement();
QDomNode noeud = racine.firstChild();
QDomElement tagQuestion;

while(!noeud.isNull())
{
  tagQuestion = noeud.toElement();
  /* est-ce la balise question ? */
  if (tagQuestion.tagName() == "question")
  {
    /* une question a traiter */
    ...
  }
  noeud = noeud.nextSibling();
}
```

Code 11: Exemple de parcours d'un fichier XML

L'élement **question** contient d'autres noeuds (c'est dû à la structure arborescente d'un document XML). Pour obtenir (et parcourir) cette liste de noeuds, on fera :

```
QDomNodeList childNodes;
childNodes = tagQuestion.childNodes();
for(unsigned int i=0;i<childNodes.length();i++)
{
   ...
}
```

Code 12: Exemple de parcours d'une liste de noeuds

Lorsqu'on parcourt cette liste de noeuds, on utilise le même principe que précédemment pour accéder aux éléments (ou balises) :

```
QDomElement tag;
QDomNode item;

for(unsigned int i=0;i<childNodes.length();i++)
{
  item = childNodes.item(i);
  tag = item.toElement();
  /* est-ce la balise libelle ? */
  if (tag.tagName() == "libelle")
  {
    ...
```

```
    qDebud() << tag.text(); /* affiche le contenu textuel de la balise */
  }
  /* est-ce la balise ... ? */
  ...
}
```

Code 13: Exemple d'accès à un élément de la liste

```
<libelle>Laquelle des expressions suivantes est un prototype de fonction ?</libelle>
```

Accès au contenu d'un élément : Pour accéder au contenu textuel (la partie comprise entre la balise ouvrante et la balise fermante) d'une balise (`<libelle>`contenu textuel`</libelle>`), on utilisera : **`tag.text()`**. Il est évidemment possible de convertir ce "texte" en entier en utilisant la méthode `toInt()` ou en réel en utilisant `toDouble()`.

```
<proposition libelle="int f(0);" />
```

Accès à un attribut d'un élément : Pour accéder à l'attribut (`libelle`) d'une balise (`proposition`), on utilisera : **`tag.attribute("libelle")`**.

Question 7. Modifier le fichier de projet (.pro) pour ajouter le support **xml**.

Question 8. Coder la méthode privée **`lireExamen()`** de la classe `QCMDialog`.

Question 9. Fabriquer l'application version 3 et tester.

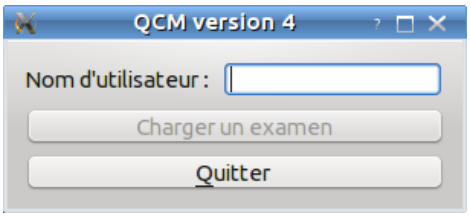
 © 2012 tv <tvaira@free.fr>

Itération 4

Dans cette dernière itération, on va ajouter deux fonctionnalités :

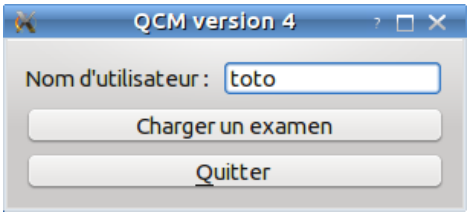
- une boîte de dialogue de lancement permettant le choix du QCM à exécuter
- un chronomètre (analogique) permettant de mesurer la durée pour répondre aux questions

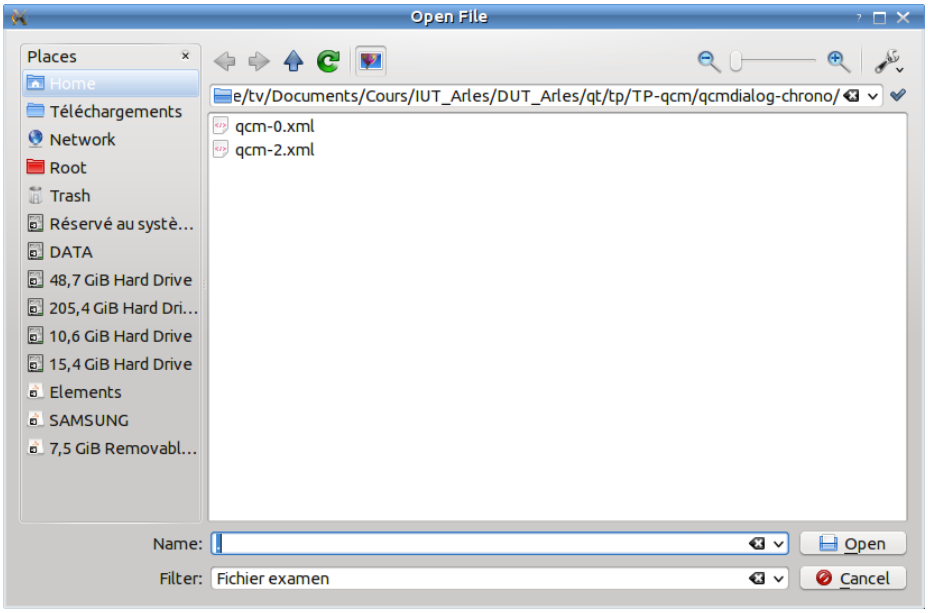
Lorsque l'utilisateur exécute l'application dans cette version, il voit apparaître une boîte de dialogue de lancement lui permettant de saisir son nom et de charger l'examen de son choix. Pour la réalisation de cette boîte, on va créer une nouvelle classe `ExamenDialog` qui héritera de `QDialog`. Cette boîte sera affichée à partir du main.



Remarque : le bouton "Charger un examen" sera valide seulement si l'utilisateur a saisi un nom.

Puis l'utilisateur clique sur le bouton "Charger un examen" et une boîte de dialogue Qt du type `QFileDialog` s'affichera lui permettant de choisir le fichier **xml** de l'examen.





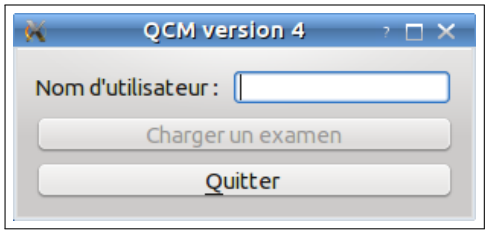
On appliquera un filtre sur les fichier d'extension xml

Après avoir cliqué sur "Open", l'examen démarre avec l'affichage d'un chronomètre. Cela revient à instancier un objet de type `QCMDialog` en lui passant en paramètre :

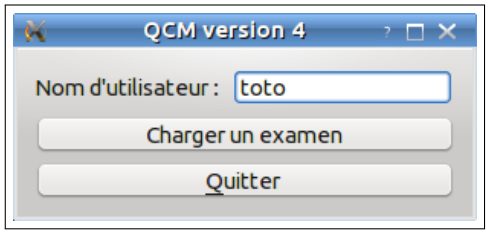
- le nom du fichier xml (`QString`)
- le nom de l'utilisateur (`QString`)

Ces deux informations permettront de créer le fichier résultat suivant : `nomExamen_nomUtilisateur.csv`. Par contre, la boîte de dialogue `QCMDialog` devra être **modale** pour bloquer l'utilisateur dans la réalisation de son QCM. Une fois terminé, on reviendra à la boîte de lancement pour "Quitter" ou "Charger un nouvel examen".

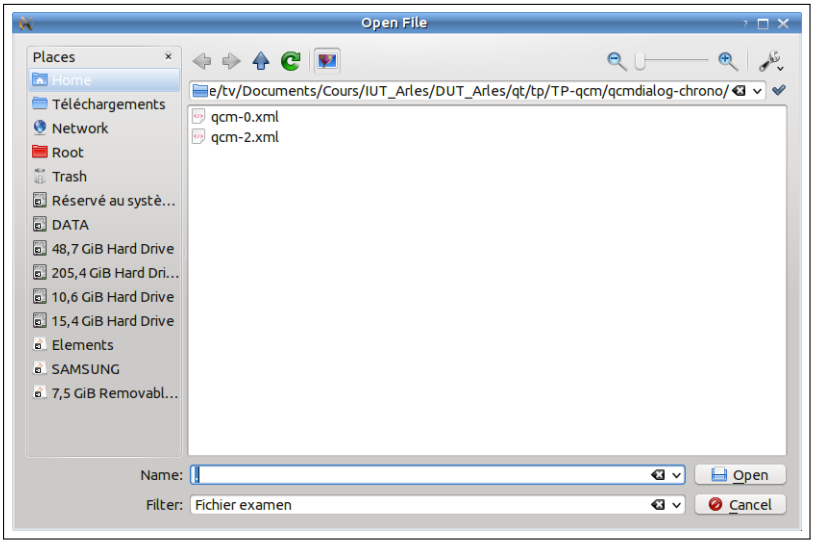
Le manuel d'utilisation pour cette version sera le suivant :



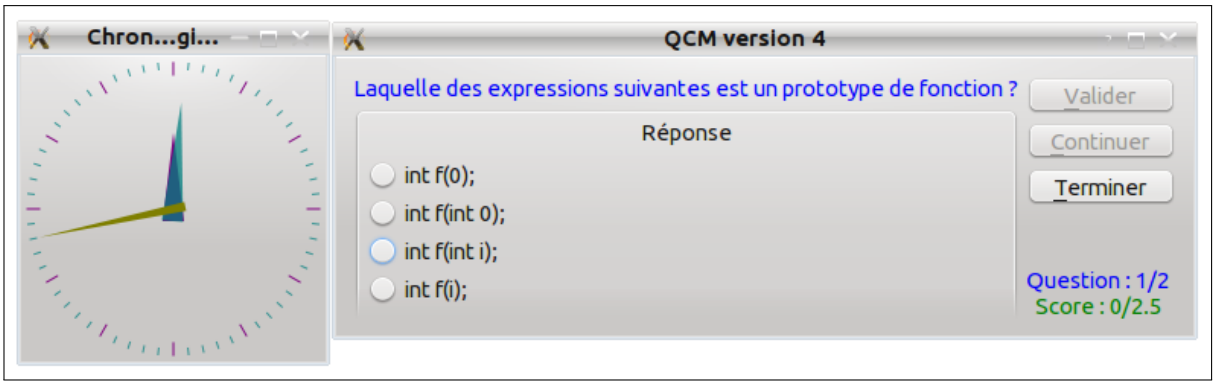
L'écran de démarrage s'affiche ...



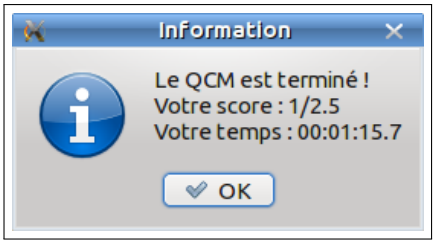
L'utilisateur saisit son nom puis il charge un examen ...



Il choisit alors le fichier Examen à charger ...

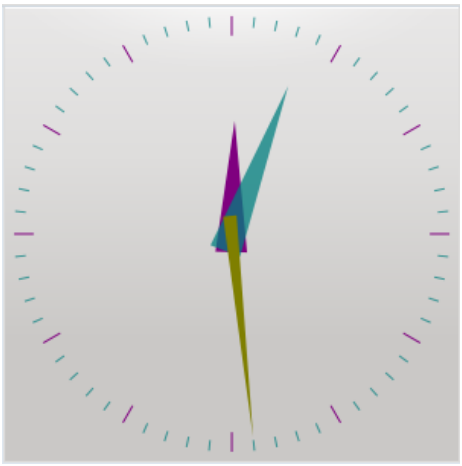


Le QCM démarre avec l'affichage du chronomètre ...



Une fois terminé, on affiche le résultat en indiquant le score et le temps.

Pour réaliser le chronomètre analogique, on se basera sur l'exemple "Analog Clock" fourni par Qt (doc.qt.nokia.com/widgets-analogclock.html). Il faudra tout de même le modifier graphiquement pour visualiser l'aiguille des secondes.



Pour la "base de temps", on utilisera un `QTimer` réglé au dixième de secondes.

Question 10. Créer la boîte de dialogue de lancement et modifier l'application existante.

Question 11. Intégrer la gestion et l'affichage du chronomètre analogique.

Question 12. Fabriquer l'application version 4 et tester.

Liens

- Documentation "Qt Reference" en français : http://qt.developpez.com/doc/
- Documentation "Qt Reference" en anglais : doc.qt.nokia.com
- Documentation "STL Containers : vector" : www.cplusplus.com/reference/stl/vector/
- Documentation `QFile` : developer.qt.nokia.com/doc/qfile.html
- Documentation `QTextStream` : developer.qt.nokia.com/doc/qtextstream.html
- Documentation "XML Processing" : developer.qt.nokia.com/doc/xml-processing.html
- XML : fr.wikipedia.org/wiki/Xml
- CSV : fr.wikipedia.org/wiki/Comma-separated_values

 © 2012 tv <tvaira@free.fr>